# Streaming multimedia files from relational database

Tomasz Rybak

Applied Systems Division
Software Departament
Faculty of Computer Science
Bialystok Technical University
rybak@ii.pb.bialystok.pl

# Outline

1. Introduction
2. Multimedia databases
   - Storage methods
   - Queries and data retrieval
3. Multimedia data as streams
   - GStreamer multimedia framework
   - Dynamic objects
   - Non-linear streams
4. Implementation
   - Description of used software
   - Implementation details
   - Problems and limitations
5. Conclusion

# Introduction

- Users posses large collections of multimedia files
- Those files are stored in file systems
- File system stores only name of the file
- Other information about file are hard to find
- Different desktop search tools store metadata in databases
    - Google Desktop Search
    - Spotlight
    - Beagle
- How to retrieve metadata and files them from the database?

# Relational databases

- Simple, atomic column type
- Small row size
- Many rows
- Relations between tables
- Normalization — avoidance of repetition

# Multimedia data

- Many different types of files
- Large files
- Internal structure
- Potentially many streams in one file

# Multimedia data in relational database

- To be consistent with 1st normal form file should be divided
- How to divide movie?
  - Into scenes?
  - Frames?
  - Pixels?
- Movie can consist of many streams
  - Video — shots from differetn cameras
  - Audio — many language versions
  - Subtitles
- Synchronisation of streams

# Storage problems

- Encoded movie takes less disk space than raw data
- When dividing movie, should we compress single frames?
- Usually unchanged movie in held database

# Storing files in database

- Binary Large OBjects — BLOB
- Special column type
- File is part of the row
- Stored inside database structure
- Special operators or functions to access data
- No need to change backup procedures
- Limit of size of stored files (usually 1 or 2GB)

# Storing files in database

- Storing files in file system
- Database only contains path to the file
- Special functions to manage files are required
- Problem with access rights
- Updating row must be joined with updating the file
- Change of backup procedures is required
- Size of file is limited only by file system

# Storing files in database

- Most solutions today uses the second method
- They store only path to file and metadata describing this file in database
- To provide consistency between files and database file monitoring tools and triggers can be used

# Queries

- Properities and metadata as attributes in query
- Precalculating properities during importing of the file
- Exact match or approximate match
- Calculate the distance of each file from the desired result in the latter case
- Return the closest ones; either choose every file closer than $\epsilon$ or N closest ones
- Should the entire movie, or only fragment be returned?

# Returning fragments of movie

- From byte to byte
  - Easy to write functions for
  - May break compression
  - May receive fragment of frame
- Only chosen frames
  - Requires knowing internal structure
  - Requires uncompressing movie, choosing frames, and recompressing it

# Streams

- Multimedia file is continous stream of data
- Either finite, or infinite (web cam)
- Streams allow to describe, analyze and modelling of networks
- Stream approach is used in describing processing of data (e.g. Unix pipes)

# GStreamer

- Open Source multimedia framework
- Very popular under Linux – used in GNOME, partially in KDE
- Based on pipelines processing multimedia streams
- Pipeline is build from fragments (plugins)
- Each fragment is responsible for simple task
- Pipeline has data bus and events bus
- Can use many threads

## Plugin

- Uses pads to communicate with other plugins
  - Sink pad
  - Source pad
- Can have any number of pads (at least one)
- Is able to process some types of data, described in capabilities
- Can generate and respond to events

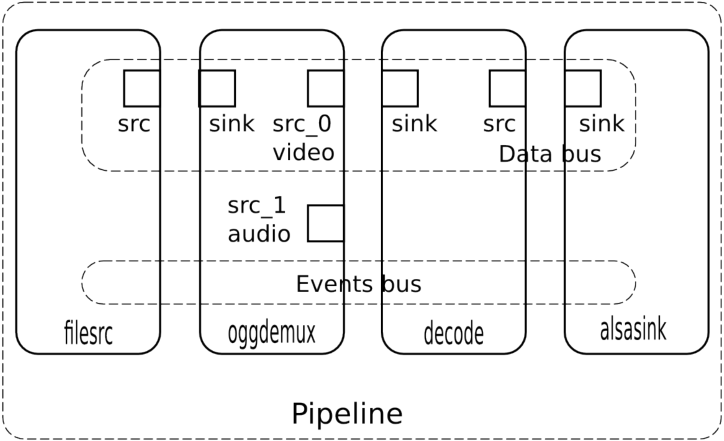

Figure: GStreamer sample pipeline

# The simplest GStreamer program in Python

```
#!/usr/bin/python

import pygst
pygst.require("0.10")
import gst
import pygtk
import gtk

class Main:
        def __init__(self):
                self.pipeline = gst.Pipeline("pipeline")
                self.audiotestsrc = gst.element_factory_make("audiotestsrc", "source")
                self.pipeline.add(self.audiotestsrc)
                self.sink = gst.element_factory_make("alsasink", "sink")
                self.pipeline.add(self.sink)
                self.audiotestsrc.link(self.sink)
                self.audiotestsrc.set_property("freq", 200);

                self.pipeline.set_state(gst.STATE_PLAYING)

start = Main()
gtk.main();
```

# Dynamic objects

- File can consist of many streams, and need many pipelines to process it
- Demultiplexing plugin can have dynamic pads, created only when needed
- Creation of new pad generates event (usually “new-pad”)

# Dynamic objects

- File can consist of many streams, and need many pipelines to process it
- Demultiplexing plugin can have dynamic pads, created only when needed
- Creation of new pad generates event (usually “new-pad”)
- Different types of files need different plugins in pipeline
- Different decoders, different sources, . . .
- GStreamer offers spacial plugins (e.g. decodebin), that contain sub-pipelines
- Those plugins are responsible for creating pipelines processing file
- They present pads with decoded data to the main pipeline

# GNonLin

- Non-linear editing of movies
    - Non-chronological order
    - Source clips remain unchanged
- GNonLin — set of plugins that offer nonlinear capabilities
- gnlcomposition contains other plugin
    - Manages decodebin plugins
    - Generates “pad-added” event
- gnlfilesource
- gnloperation

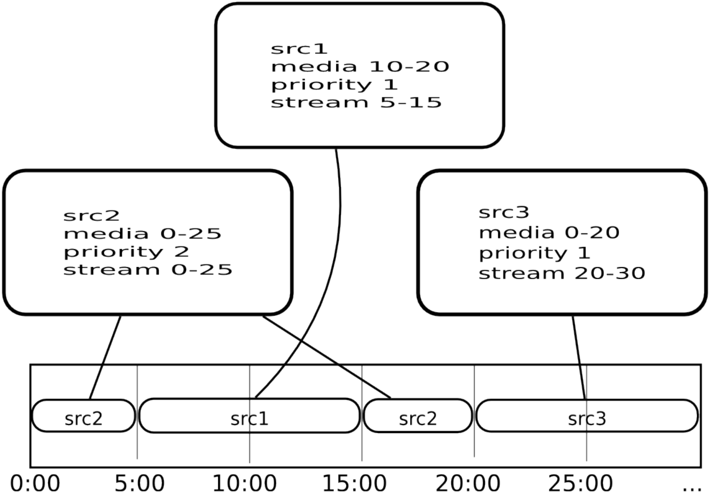

Figure: GNonLin media stream

# Software

- PostgreSQL as database
- GStreamer as multimedia framework
- Programming language Python

# System description

- Database contains description of audio and video files
- Client chooses file to receive
- Database server sends streams to the client
- Client receives and displays stream
- Server and client can be on different machines

# Script used for populating database

```
#! /bin/sh

for name in $1/*
do
        command="INSERT INTO movies (name, path) VALUES (’"‘basename "$name" .avi‘"’, ’"file://$name"’);"
        echo $command
        psql -p 5432 -c "$command"
done
```

# Used plugins

- Server must behave as a source and client as a sink
- Creation of custom plugins would be to complicated
- GStreamer offers networking plugins: tcpserversink and tcpclientsrc
- They allow to transfer one stream
- Streams were encoded in OGG file
- OGG can store many concurrent streams

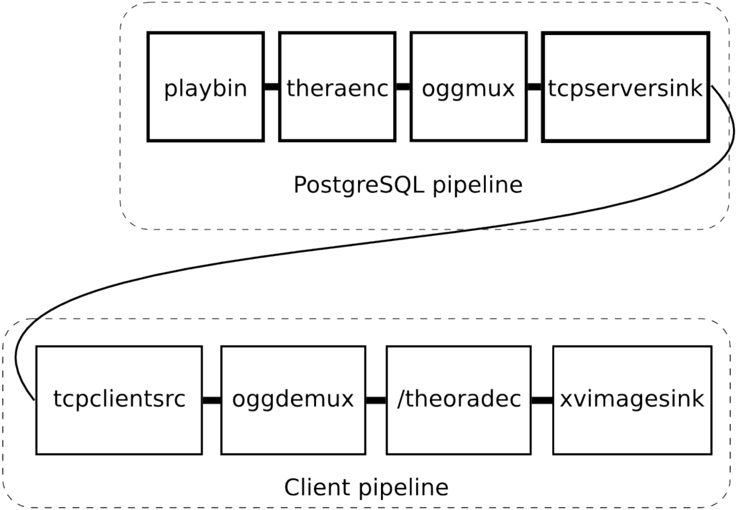

Figure: Implemented video pipelines

# Video server

```
#! /usr/bin/python

import pygst
pygst.require("0.10")
import gst
import gobject

class Main:
        def new_pad(self, dbin, pad):
                if not "video" in pad.get_caps().to_string(): return
                pad.link(self.fd.get_pad("sink"))
        def __init__(self):
                self.pipeline = gst.Pipeline("pipes")

                fs = gst.element_factory_make("filesrc", "fs")
                fs.set_property("location", "/tmp/video.avi")
                self.pipeline.add(fs)

                ad = gst.element_factory_make("avidemux", "ad")
                self.pipeline.add(ad)
                fs.link(ad)
                ad.connect("pad-added", self.new_pad)
                self.fd = gst.element_factory_make("ffdec_mpeg4", "fd")
                self.pipeline.add(self.fd)
                ve = gst.element_factory_make("theoraenc", "ve")
                self.pipeline.add(ve)
                self.fd.link(ve)
                om = gst.element_factory_make("oggmux", "om")
                self.pipeline.add(om)
                ve.link(om)
```

# Video client

```
#! /bin/sh

gst-launch tcpclientsrc host="127.0.0.1" port="1234" ! \
queue ! oggdemux ! theoradec ! \
xvimagesink
```

# PostgreSQL functions

```
CREATE TABLE movies
(
        id SERIAL,
        name TEXT,
        path TEXT
);

GRANT ALL PRIVILEGES ON movies TO tomus;
GRANT ALL PRIVILEGES ON SEQUENCE movies_id_seq TO tomus;

CREATE TYPE stream_info AS (id INTEGER, caps TEXT, codec TEXT, streamtype TEXT);

CREATE OR REPLACE FUNCTION get_stream_info(filename TEXT)
RETURNS SETOF stream_info
VOLATILE RETURNS NULL ON NULL INPUT SECURITY DEFINER
LANGUAGE ’plpythonu’ AS
$$
import plpy
import time
import pygst
pygst.require("0.10")
import gst

class stream_info:
        def __init__(self, streams):
                self.streams = streams
                self.i = -1
        def __iter__(self):
                return self
        def next(self):
                self.i += 1
```

# SQL functions

get_stream_info(filename) returns information about all streams contained in file

get_stream(filename, host, port, audio, video) creates pipeline returning requested audio and video stream on requested address and port

# Python client

```
#! /usr/bin/python

import pygst
pygst.require("0.10")
import gst
import pygtk
import gtk
import gtk.glade
import psycopg2
import time

class Main:
        def __init__(self):
                self.connection = psycopg2.connect("dbname=tomus port=5432")
                self.tree = gtk.glade.XML("client.glade", "client")
                signals = {
                        "on_play_clicked" : self.OnPlay,
                        "on_stop_clicked" : self.OnStop,
                        "on_quit_clicked" : self.OnQuit,
                }
                self.tree.signal_autoconnect(signals)

                self.pipeline = gst.Pipeline("pipes")

                tc = gst.element_factory_make("tcpclientsrc", "tc")
                tc.set_property("host", "127.0.0.1")
                tc.set_property("port", 1234)
                self.pipeline.add(tc)
                od = gst.element_factory_make("oggdemux", "od")
                self.pipeline.add(od)
                od.connect("pad-added", self.OnPad)
```

# Problems, limitations

- GNonLin does not work with networking plugins
- Two ports opened in the server
  - SQL connection
  - multimedia stream
- Only one stream of each type (audio/video) is transmitted
- Decoding and encoding into Ogg/Theora+Vorbis takes much processor power
  - Recode into OGG during importing

# Future ideas

- Processing of files inside database
- Custom GStreamer plugins to retrieve metadata
- Query language using that metadata
- Custom types, indexes, etc.
- Using GNonLin to join movies being result of one query.
- Variable media quality (similar to Move framework by Drew Major, http://www.movenetworks.com/)

# Questions

Questions?
Thank you for your attention.
Tomasz Rybak
rybak@ii.pb.bialystok.pl